Apple II
Technical Notes

Developer Technical Support

Apple IIGS
#51: How to Avoid Running Out of Memory

Revised by: Dave Lyons May 1992
Written by: Eric Soldan January 1989

This Technical Note discusses handling nearly-out-of-memory situations when working with the IIGS tools.
Changes since September 1990: Added discussion of an Out-of-memory routine problem fixed in System 6.0.

Introduction

Running out of memory is a concern for most every application. Working with the Toolbox makes monitoring this situation a little more difficult since your application is not the only one allocating memory.

Low-level toolbox functions (for example, QuickDraw II calls) require that a 16K block of memory be allocatable, while high-level routines (for example, the Window Manager) require that a 32K block of memory be allocatable. Apple does not guarantee that toolbox functions behave reasonably if there is less memory available, and the tools are not stress-tested with less than the minimum required memory available.

Since the toolbox assumes reasonable memory-allocation requests succeed, just waiting for an out-of-memory error is **not** adequate memory management. To make your application work reliably in low-memory situations, you need a method of ensuring that the toolbox gets memory when it needs it. This Note describes two approaches.

How Much Memory Can Be Allocated

There's no way to tell how much memory can be allocated without actually trying to allocate it.

`MaxBlock` tells you the size of the largest single free block, but this doesn't take into account purgeable blocks, compaction, and out-of-memory routines (see *Apple IIGS Toolbox Reference*, volume 3). `FreeMem` and `RealFreeMem` cannot tell you how badly fragmented the memory is, and they do not take into account out-of-memory routines.

A Suggested Method

A method of checking for a nearly-out-of-memory condition is to have your own purgeable handle just for this task. If the handle has not been purged, then you have plenty of memory for the toolbox, and in the worst case, the toolbox purges your handle if it needs the RAM.

The less often your purgeable handle gets purged, the better performance you get in nearly-out-of-memory situations. Therefore, you should arrange for other purgeable memory, not necessarily belonging to your application, to be purged **before** your handle. For example, you want dormant applications to be purged, rather than having your handle get repeatedly purged and reallocated. So the purge level of this handle should be one.

The check to see if a handle has been purged is very fast. If it has been purged, you have to try to reallocate it. Reallocating a handle is not a fast process, so the fewer times the handle is purged, the faster the check is and the better your performance. Unless you are in a nearly-out-of-memory situation, the handle should not be purged at all, and you should have virtually no overhead for this process.

This technique can be implemented as follows:

```
appStart
;
; Somewhere at start, create a purgeable handle of size N,
; called "loMemHndl", purge level 1.
;
                rts

******************
;
; Here's an example of checking for nearly-out-of-memory:
;
                jsr    preCheckLoMem
                bcc    goForIt
                bcs    HandleError        ;Handle errors appropriately.
goForIt         (_ToolboxCall[s])         ;Make as many as needed.
;
; Here you can make your toolbox calls.  Since you prechecked
; for nearly-out-of-memory conditions, you should have no memory
; errors at this point.
;
; You could also check after calls, as shown here:
;
                (_ToolboxCall)
                jsr    checkLoMem         ;Call this to see if low.
                bcc    noError
                bcs    HandleError        ;Take care of errors.

noError         jsr    lifeIsGood
                .
                .
                .
                rts
```

```
******************
;
; Here are some sample routines to check for the nearly-out-of-
; memory condition.
;
checkLoMem      bcs    retErr
preCheckLoMem   lda    [loMemHndl]
                ldy    #2
                ora    [loMemHndl],y
                beq    gotPurged
                lda    #0
                clc
                rts
gotPurged       (Try reallocating it into loMemHndl, purge level 1.)
                (If you can't, you will get a $0201 error.  You may wish to
                 return the $201 error, or you may wish to change it into
                 your own error code.)
;
retErr          rts                     ;This is a single exit point
                                        ;whether errors were present
                                        ;or not.
```

You can determine the size of this purgeable handle, but like determining what size stack is adequate for an application, there is no single "right" answer. There are different considerations for size of the purgeable handle for each application, and these may change during the development process. Use your best judgement, keeping in mind that high-level toolbox routines require a 32K block.

An Alternative

For better control over when your handle is purged or disposed, you can write an out-of-memory routine as described in the Memory Manager chapter of *Apple IIGS Toolbox Reference*, volume 3. Out-of-memory routines have the opportunity to free up memory before or after the Memory Manager attempts to purge purgeable handles, and this manual contains a sample of such a routine.

Note: If your Out-of-memory routine frees up memory on the second pass, there is a problem with the Memory Manager in System Software 5.0 through 5.0.4 that may affect you. If your routine frees enough bytes on the second pass, but the Memory Manager still cannot complete the request it is working on, it can hang for a couple of minutes and then crash. This is fixed in System 6.0.

Further Reference:

- *Apple IIGS Toolbox Reference*, Volumes 1-3